Least Privilege

Sean Barnum, Cigital, Inc. [vita[3]]
Michael Gegick, Cigital, Inc. [vita[4]]

2005-09-14

L4 / D/P, L[5]

Only the minimum necessary rights should be assigned to a subject that requests access to a resource and should be in effect for the shortest duration necessary (remember to relinquish privileges). Granting permissions to a user beyond the scope of the necessary rights of an action can allow that user to obtain or change information in unwanted ways. Therefore, careful delegation of access rights can limit attackers from damaging a system.

Detailed Description Excerpts

According to Saltzer and Schroeder [Saltzer 75] in "Basic Principles of Information Protection," page 9:

> Least privilege: Every program and every user of the system should operate using the least set of privileges necessary to complete the job. Primarily, this principle limits the damage that can result from an accident or error. It also reduces the number of potential interactions among privileged programs to the minimum for correct operation, so that unintentional, unwanted, or improper uses of privilege are less likely to occur. Thus, if a question arises related to misuse of a privilege, the number of programs that must be audited is minimized. Put another way, if a mechanism can provide "firewalls," the principle of least privilege provides a rationale for where to install the firewalls. The military security rule of "need-to-know" is an example of this principle.

According to Bishop [Bishop 03] in Chapter 13, "Design Principles," Section 13.2.1, "Principle of Least Privilege," pages 343-344:[9]

> This principle restricts how privileges are granted.
>
> Definition 13-1. The Principle of Least Privilege states that a subject should be given only those privileges needed for it to complete its task.
>
> If a subject does not need an access right, the subject should not have that right. Further, the function of the subject (as opposed to its identity) should control the assignment of rights. If a specific action requires that a subject's access rights be augmented, those extra rights should be relinquished immediately upon completion of the action. This is the analogue of the "need to know" rule: if the subject does not need access to an object to perform its task, it should not have the right to access that object. More precisely, if a subject needs to append to an object, but not to alter the information already contained in the object, it should be given append rights and not write rights.
>
> In practice, most systems do not have the needed granularity of privileges and permissions to apply this principle precisely. The designers of security mechanisms then apply this principle as best they can. In such systems, the consequences of security problems are often more severe than the consequences on systems which adhere to this principle.
>
> This principle requires that processes should be confined to as small a protection domain as possible.
>
> Example 1

3. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/35-BSI.html (Barnum, Sean)
4. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/345-BSI.html (Gegick, Michael)
9. This excerpt is from the book *Computer Security: Art and Science*, written by Matt Bishop, ISBN 0-201-44099-7, copyright 2003. All rights reserved. It is reprinted with permission from Addison-Wesley Professional.

The UNIX operating system does not apply access controls to the user root. That user can terminate any process and read, write, or delete any file. Thus, users who create back-ups can also delete files. The administrator account on Windows has the same powers.

Example 2

A mail server accepts mail from the Internet, and copies the messages into a spool directory; a local server will complete delivery. It needs rights to access the appropriate network port, to create files in the spool directory, and to alter those files (so it can copy the message into the file, rewrite the delivery address if needed, and add the appropriate "Received" lines). It should surrender the right to access the file as soon as it has completed writing the file into the spool directory, because it does not need to access that file again. The server should not be able to access any user's files, or any files other than its own configuration files.

According to Viega and McGraw [Viega 02] in Chapter 5, "Guiding Principles for Software Security," in "Principle 4: Follow the Principle of Least Privilege" from pages 100-103:[10]

The principle of least privilege states that only the minimum access necessary to perform an operation should be granted, and that access should be granted only for the minimum amount of time necessary. (This principle was introduced by Saltzer and Schroeder.[11])

When you give out access to parts of a system, there is always some risk that the privileges associated with that access will be abused.

This problem is starting to become common in security policies that ship with products intended to run in a restricted environment. Some vendors offer applications that work as Java applets. Applets usually constitute mobile code, which a web browser treats with suspicion by default. Such code is run in a sandbox, where the behavior of the applet is restricted based on a security policy that a user sets. Vendors rarely practice the principle of least privilege when they suggest a policy to use with their code, because doing so would take a lot of effort on their part. It's far easier to just ship a policy that says, "let my code do anything at all." People will generally install vendor-supplied security policies, maybe because they trust the vendor, or maybe because it's too much hassle to figure out what security policy does the best job of minimizing the privileges that must be granted to the vendor's application.

Laziness often works against the principle of least privilege. Don't let that happen in your code.

Example 1

For example, let's say you were to go on vacation, and give a friend the key to your home, just to feed pets, collect mail, etc. While you may trust a friend, there is always the possibility that there will be a party in your house without your consent, or that something else will happen that you don't like. Whether or not you trust your friend, there's really no need to put yourself at risk by giving more access than necessary. For example, if you don't have pets, but only needed a friend to occasionally pick up our mail, you should relinquish only the mailbox key. While your friend might find a good way to abuse that privilege, at least you don't have to worry about the possibility of additional abuse. If you give out the house key unnecessarily, all that changes.

Similarly, if you do get a house sitter while you're on vacation, you aren't likely to let that person keep your keys when you're not on vacation. If you do, you're setting yourself up for additional risk. Whenever a key to your house is out of your control, there's a risk of that key getting duplicated. If there's a key outside your control, and you're not home, then there's the risk that the key is being used to enter your house. Any length of time when someone has your key and is not being supervised by you constitutes a window of time in which you are vulnerable to an attack. You want to keep such windows of vulnerability as short as possible, in order to minimize your risks.

10. All rights reserved. It is reprinted with permission from Addison-Wesley Professional.
11. Saltzer, Jerome H. & Schroeder, Michael D. "The Protection of Information in Computer Systems." *Proceedings of the IEEE 63*, 9 (September 1975): 1278-1308.

Example 2

Another good real-world example appears in the security clearance system of the U.S. government; in particular with the notion of "need to know". If you have clearance to see any secret document whatsoever, you still can't demand to see any secret document that you know exists. If you could, it would be very easy to abuse the security clearance level. Instead, people are only allowed to access documents that are relevant to whatever task they are supposed to perform.

Example 3

Some of the most famous violations of the principle of least privilege exist in UNIX systems. For example, in UNIX systems, root privileges are necessary to bind a program to a port number less than 1024. For example, to run a mail server on port 25, the traditional SMTP port, a program needs the privileges of the root user. However, once a program has set up shop on port 25, there is no compelling need for it to ever use root privileges again. A security-conscious program relinquishes root privileges as soon as possible, and will let the operating system know that it should never require those privileges again in this execution (see Chapter 8 [in *Building Secure Software*] for a discussion of privileges). One large problem with many e-mail servers is that they don't give up their root permissions once they grab the mail port (Sendmail is a classic example). Therefore, if someone ever finds a way to trick such a mail server into doing something nefarious, it will be able to get root. So if a malicious attacker were to find a suitable stack overflow in Sendmail (see Chapter 7 [in *Building Secure Software*]), that overflow could be used to trick the program into running arbitrary code as root. Given root permission, anything valid the attacker tries will succeed. The problem of relinquishing privilege is especially bad in Java, since there is no operating-system independent way to give up permissions.

Example 4

Another common scenario involves a programmer who may wish to access some sort of data object, but only needs to read from the object. Let's say the programmer actually requests more privileges than necessary, for whatever reason. Programmers do this to make life easier. For example, one might say, "Someday I might need to write to this object, and it would suck to have to go back and change this request." Insecure defaults might lead to a violation here, too. For example, there are several calls in the Windows API for accessing objects that grant all access if you pass "0" as an argument. In order to get something more restrictive, you'd need to pass a bunch of flags (OR'd together). Many programmers will just stick with the default, as long as it works, since that's easiest.

According to Howard and LeBlanc [Howard 02] in Chapter 3, "Security Principles to Live By," in "Use Least Privilege" from pages 60-61:

> All applications should execute with the least privilege to get the job done and no more. I often analyze products that must be executed in the security context of an administrative account--or, worse, as a service running as the Local System account-when, with some thought, the product designers could have not required such privileged accounts. The reason for running with least privilege is quite simple. If a security vulnerability is found in the code and an attacker can inject code into your process, make the code perform sensitive tasks, or run a Trojan horse or virus, the malicious code will run with the same privileges as the compromised process. If the process is running as an administrator, the malicious code runs as an administrator. This is why we recommend people do not run as a member of the local administrators group on their computers, just in case a virus or some other malicious code executes.
>
> Go on, admit it: you're logged on to your computer as a member of the local administrators group, aren't you" I'm not. I haven't been for over three years, and everything works fine. I write code, I debug code, I send e-mail, I sync with my Pocket PC, I create documentation for an intranet site, and do myriad other things. To do all this, you don't need admin rights, so why run as an admin? (I will

admit that when I build a new computer I add myself to the admin group, install all the applications I need, and then promptly remove myself.)

When you create your application, write down what resources it must access and what special tasks it must perform. Examples of resources include files and registry data; examples of special tasks include the ability to log user accounts on to the system, debug processes, or backup data. Often you'll find you do not require many special privileges or capabilities to get any tasks done. Once you have a list of all your resources, determine what might need to be done with those resources. For example, a user might need to read and write to the resources but not create or delete them. Armed with this information, you can determine whether the user needs to run as an administrator to use your application. The chances are good that she does not.

A common use of least privilege again involves banks. The most valued part of a bank is the vault, but the tellers do not generally have access to the vault. That way an attacker could threaten a teller to access the vault, but the teller simply won't know how to do it.

For a humorous look at the principle of least privilege, refer to "If we don't run as admin, stuff breaks" in Appendix B [in *Writing Secure Code*], "Ridiculous Excuses We've Heard." Also, see Chapter 7 [in *Writing Secure Code*] for a full account of how you can often get around requiring dangerous privileges.

Tip: If your application fails to run unless the user (or service process identity) is an administrator or the system account, determine why. Chances are good that elevated privileges are unnecessary.

According to NIST [NIST 01] in Section 3.3, "IT Security Principles," from page 16:

Implement least privilege.

The concept of limiting access, or "least privilege," is simply to provide no more authorizations than necessary to perform required functions. This is perhaps most often applied in the administration of the system. Its goal is to reduce risk by limiting the number of people with access to critical system security controls; i.e., controlling who is allowed to enable or disable system security features or change the privileges of users or programs. Best practice suggests it is better to have several administrators with limited access to security resources rather than one person with "super user" permissions.

Consideration should be given to implementing role-based access controls for various aspects of system use, not only administration. The system security policy can identify and define the various roles of users or processes. Each role is assigned those permissions needed to perform its functions. Each permission specifies a permitted access to a particular resource (such as "read" and "write" access to a specified file or directory, "connect" access to a given host and port, etc.). Unless a permission is granted explicitly, the user or process should not be able to access the protected resource.

According to Schneier [Schneier 00] in "Security Processes":

Limit Privilege.

Don't give any user more privileges than he absolutely needs to do his job. Just as you wouldn't give a random employee the keys to the CEO's office, don't give him a password to the CEO's files.

What Goes Wrong

According to McGraw and Viega [McGraw 03]:[14]

Little problems can become big problems when they happen in privileged sections of code (think SUID code or code that must be run as Administrator to work). Sometimes they're introduced by installation or configuration—something that's impossible for a developer to control. For example, users commonly install a Web server and run it in a real user process space, without creating a

14. All rights reserved. It is reprinted with permission from CMP Media LLC.

nonprivileged "nobody" as the target. Also consider that Solaris SUID binaries can be run without an s-bit set, introducing unacceptable security risk.

Even if you do carefully dole out privilege, relinquishing the privilege isn't always a trivial task. Nevertheless, do it whenever you can.

References

[Bishop 03]	Bishop, Matt. *Computer Security: Art and Science*. Boston, MA: Addison-Wesley, 2003.
[Howard 02]	Howard, Michael & LeBlanc, David. *Writing Secure Code, 2nd ed*. Redmond, WA: Microsoft Press, 2002.
[McGraw 03]	McGraw, Gary & Viega, John. "Keep It Simple." *Software Development.* CMP Media LLC, May, 2003.
[NIST 01]	NIST. *Engineering Principles for Information Technology Security*. Special Publication 800-27. US Department of Commerce, National Institute of Standards and Technology, 2001.
[Saltzer 75]	Saltzer, Jerome H. & Schroeder, Michael D. "The Protection of Information in Computer Systems," 1278-1308. *Proceedings of the IEEE 63*, 9 (September 1975).
[Schneier 00]	Schneier, Bruce. "The Process of Security[16]." *Information Security Magazine*, April, 2000.
[Viega 02]	Viega, John & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison-Wesley, 2002.

Cigital, Inc. Copyright

1. mailto:copyright@cigital.com